The dataset consists of 614 rows and 13 columns, with each row representing a specific loan applicant and each column representing a different attribute or characteristic of the applicants.

Let's break down the columns and their potential relevance to a data science project:

1. Loan_ID: A unique identifier for each loan applicant. It may not have much significance in the data analysis itself but can be useful for tracking and referencing specific applicants.
2. Gender: Indicates the gender of the loan applicant. This attribute could be analyzed to explore potential gender-based patterns or biases in loan approval.
3. Married: Indicates whether the applicant is married or not. It can be used to investigate the impact of marital status on loan approvals.
4. Dependents: Indicates the number of dependents the applicant has. This attribute may help understand how the number of dependents affects loan approval decisions.
5. Education: Indicates the educational background of the applicant, distinguishing between "Graduate" and "Not Graduate." Analyzing this attribute could provide insights into the relationship between education level and loan approval.
6. Self_Employed: Indicates whether the applicant is self-employed or not. This attribute can be examined to understand the impact of self-employment on loan approvals.
7. ApplicantIncome: Represents the income of the applicant. This numerical attribute can be analyzed to assess the relationship between income level and loan approval.
8. CoapplicantIncome: Represents the income of the co-applicant, if any. Similar to ApplicantIncome, this attribute can be used to explore the combined income of applicants and its influence on loan approval.
9. LoanAmount: Indicates the amount of the loan requested by the applicant. This numerical attribute can be analyzed to understand the loan amounts preferred by applicants and their relationship with loan approval.
10. Loan_Amount_Term: Represents the term (in months) of the loan requested. This attribute may provide insights into the preferred loan durations among applicants.
11. Credit_History: Indicates the credit history of the applicant, represented by a binary value (0 or 1). This attribute can be crucial in assessing the creditworthiness of applicants and its impact on loan approval.
12. Property_Area: Indicates the type of property area where the applicant resides. This attribute can be explored to understand if there are any regional variations in loan approval rates.
13. Loan_Status: Represents the final status of the loan application, whether it was approved or not. This column can be considered the target variable for predictive modeling, where various attributes are used to predict loan approval outcomes.

In a data science project based on this dataset, you would likely perform various tasks such as data cleaning, exploratory data analysis, feature engineering, and potentially building predictive models. The specific goals of the project would depend on the context and the problem statement. Some possible objectives could be:

1. Analyzing the factors that influence loan approval decisions

1. Analyzing the factors that influence loan approval decisions.
2. Predicting loan approval outcomes based on applicant information.
3. Identifying patterns or biases in loan approvals based on gender, marital status, or other factors.
4. Understanding the relationship between income, loan amount, and loan approval.
5. Assessing the impact of credit history on loan approval rates.
6. Investigating regional variations in loan approval rates based on property area..

Data Cleaning

In [434]:
```
import pandas as pd
```

In [435]:
```
df = pd.read_csv('loan data.csv')
```

In [436]:
```
df
```

Out[436]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amou
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	
...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	

614 rows × 13 columns

In [437]:
```
df.shape
```

Out[437]:
```
(614, 13)
```

In [438]:
```
df.isna().sum()
```

Out[438]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [439]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [440]:
```
df.columns
```

Out[440]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [441]:
```
len(df)
```

Out[441]:
```
614
```

Fill missing numerical values

In [442]:
```
for label, content in df.items():
    if pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
```

checking numerical value with null values

In [443]:
```
for label, content in df.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

```
LoanAmount
Loan_Amount_Term
Credit_History
```

fill numerical rows with the median

In [444]:
```
for label, content in df.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():

            # add binary column which tells us if the data was missing
            df[label+'_missing'] = pd.isnull(content)

            # fill missing values with median
            df[label] = content.fillna(content.median())
            print(label)
```

```
LoanAmount
Loan_Amount_Term
Credit_History
```

In [445]:
```
df.isna().sum()
```

Out[445]:
```
Loan_ID                    0
Gender                    13
Married                    3
Dependents                15
Education                  0
Self_Employed             32
ApplicantIncome            0
CoapplicantIncome          0
LoanAmount                 0
Loan_Amount_Term           0
Credit_History             0
Property_Area              0
Loan_Status                0
LoanAmount_missing         0
Loan_Amount_Term_missing   0
Credit_History_missing     0
dtype: int64
```

In [446]:
```
df.head()
```

Out[446]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

Fill missing object values

In [447]:
```
df = df.fillna( ' ')
```

In [448]:
```
df.head()
```

Out[448]:

oanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	LoanAmount_missing	Loan_Amount_Term_missing	Credit_His
128.0	360.0	1.0	Urban	Y	True	False	
128.0	360.0	1.0	Rural	N	False	False	
66.0	360.0	1.0	Urban	Y	False	False	
120.0	360.0	1.0	Urban	Y	False	False	
141.0	360.0	1.0	Urban	Y	False	False	

In [449]:
```
df.isna().sum()
```

Out[449]:
```
Loan_ID                    0
Gender                     0
Married                    0
Dependents                 0
Education                  0
Self_Employed              0
ApplicantIncome            0
CoapplicantIncome          0
LoanAmount                 0
Loan_Amount_Term           0
Credit_History             0
Property_Area              0
Loan_Status                0
LoanAmount_missing         0
Loan_Amount_Term_missing   0
Credit_History_missing     0
dtype: int64
```

Exploratory Data Analysis

In [450]:
```
# importing dependencies

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import plotly.express as px
```

In [451]:
```
df.describe()
```

Out[451]:

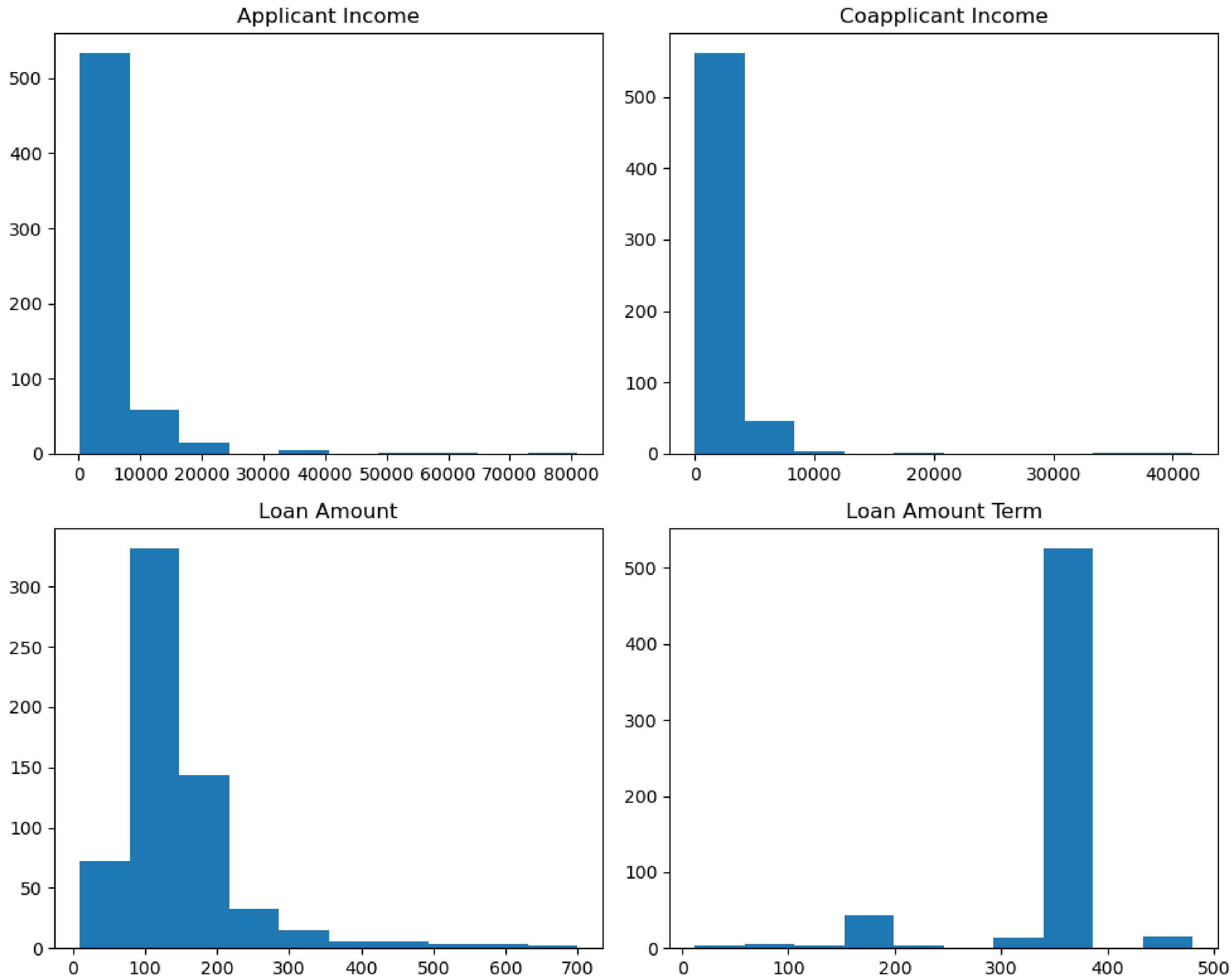
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	614.000000	614.000000
mean	5403.459283	1621.245798	145.752443	342.410423	0.855049
std	6109.041673	2926.248369	84.107233	64.428629	0.352339
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.250000	360.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

In [452]:
```
df.head()
```

Out[452]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [453]:
```
# Visualize distributions and relationships
# Create subplots with adjusted spacing
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10, 8), tight_layout=True)

# Plot histograms on each subplot
axes[0, 0].hist(df["ApplicantIncome"])
axes[0, 0].set_title("Applicant Income")

axes[0, 1].hist(df["CoapplicantIncome"])
axes[0, 1].set_title("Coapplicant Income")

axes[1, 0].hist(df["LoanAmount"])
axes[1, 0].set_title("Loan Amount")

axes[1, 1].hist(df["Loan_Amount_Term"])
axes[1, 1].set_title("Loan Amount Term")

# Display the plots
plt.show()
```


Applicant Income
0
100
200
300
400
500
0 10000 20000 30000 40000 50000 60000 70000 80000
Coapplicant Income
0
100
200
300
400
500
0 10000 20000 30000 40000
Loan Amount
0
50
100
150
200
250
300
0 100 200 300 400 500 600 700
Loan Amount Term
0
100
200
300
400
500
0 100 200 300 400 500

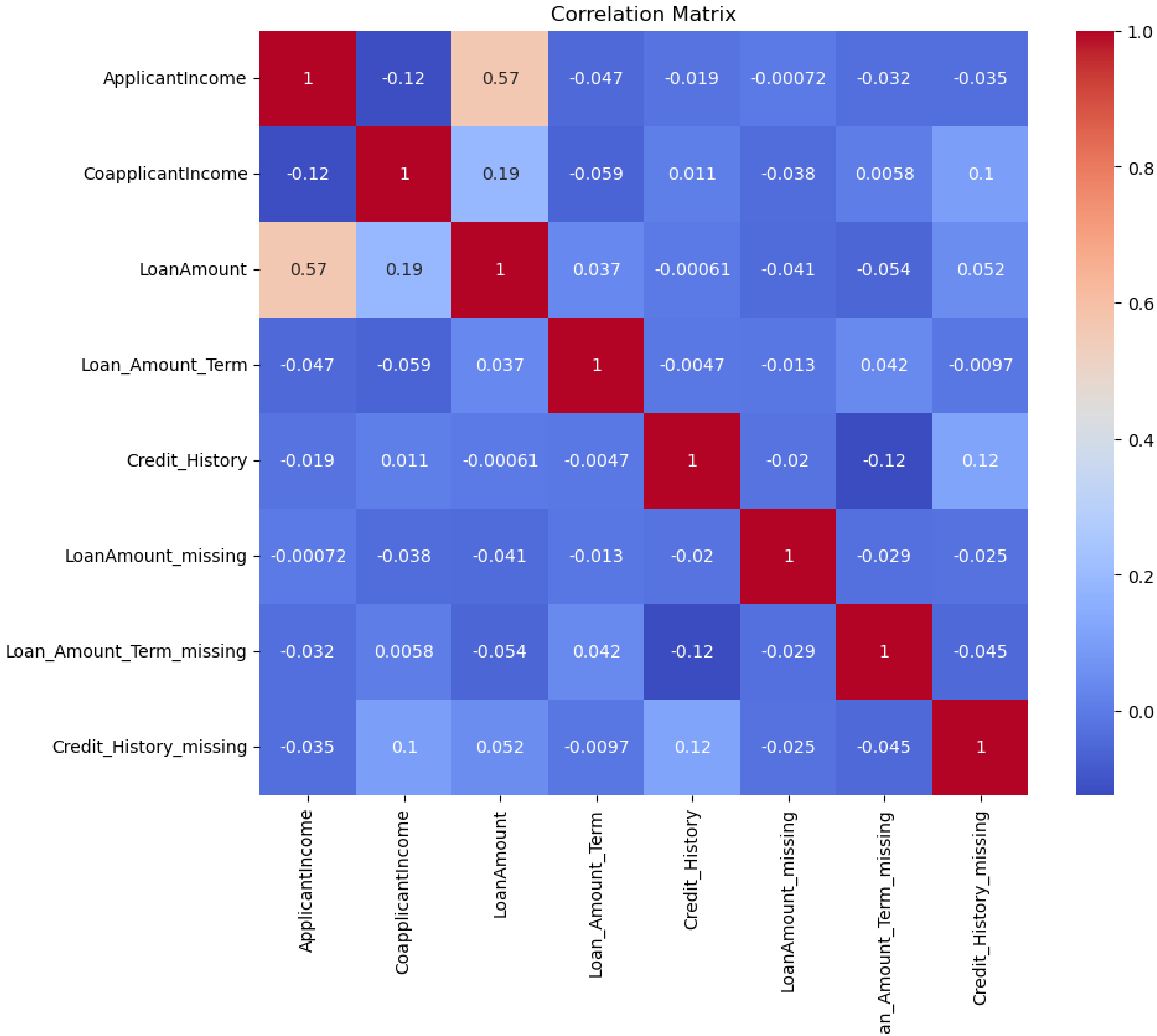
In [454]:
```
# Calculate correlation between variables
correlation_matrix = df.corr()
```

In [455]:
```
correlation_matrix
```

Out[455]:

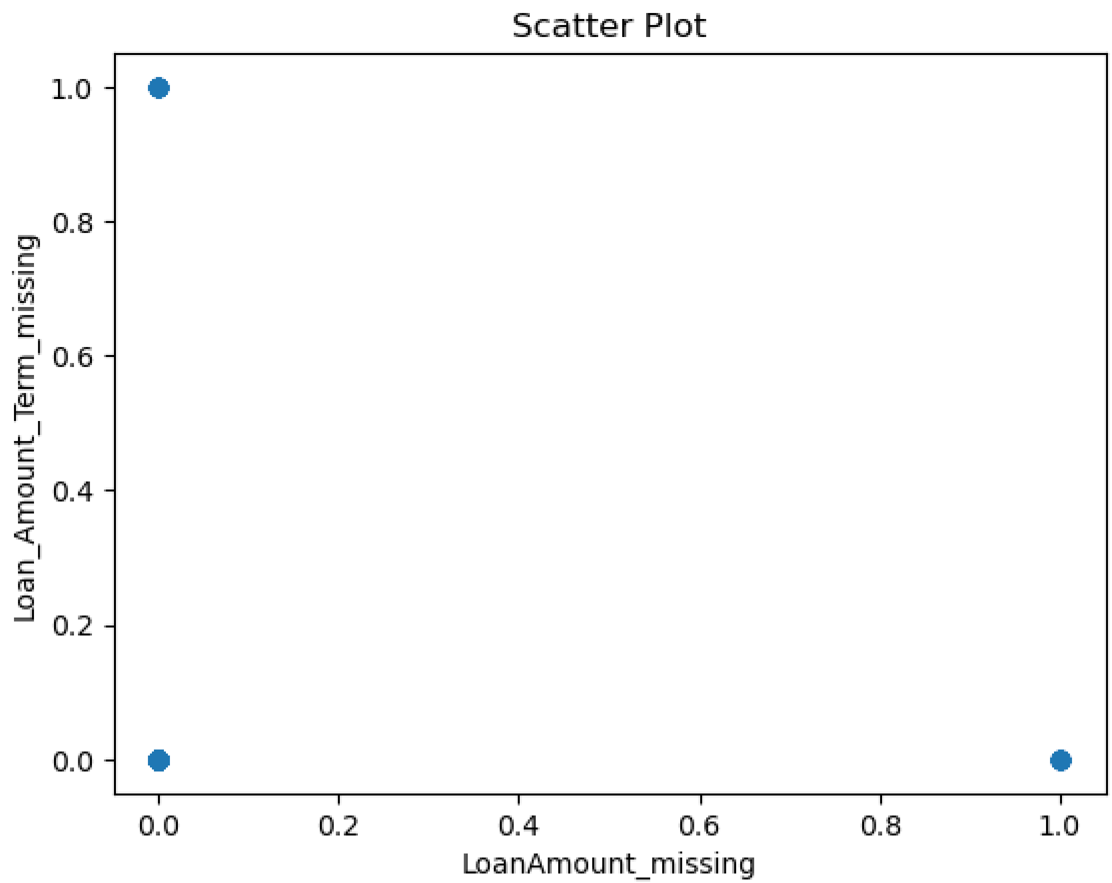
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	LoanAmount_missing
ApplicantIncome	1.000000	-0.116605	0.565181	-0.046531	-0.018615	-0.000718
CoapplicantIncome	-0.116605	1.000000	0.189218	-0.059383	0.011134	-0.037945
LoanAmount	0.565181	0.189218	1.000000	0.036960	-0.000607	-0.040722
Loan_Amount_Term	-0.046531	-0.059383	0.036960	1.000000	-0.004705	-0.012663
Credit_History	-0.018615	0.011134	-0.000607	-0.004705	1.000000	-0.020187
LoanAmount_missing	-0.000718	-0.037945	-0.040722	-0.012663	-0.020187	1.000000
Loan_Amount_Term_missing	-0.031836	0.005756	-0.053690	0.041737	-0.123061	-0.029447
Credit_History_missing	-0.034651	0.104297	0.051967	-0.009668	0.122592	-0.025359

In [456]:
```
# Calculate correlation matrix
correlation_matrix = df.corr()

# Plot correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
```

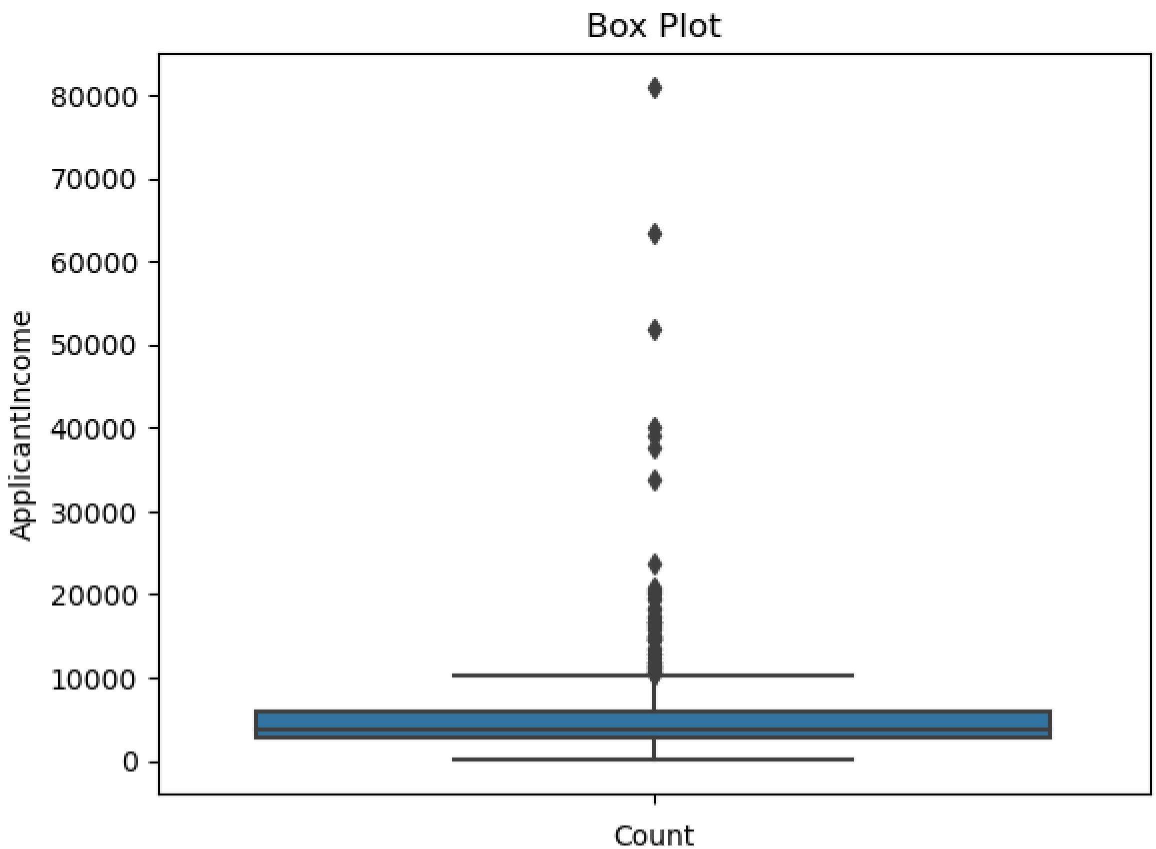

Correlation Matrix
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
LoanAmount_missing
Loan_Amount_Term_missing
Credit_History_missing
1 -0.12 0.57 -0.047 -0.019 -0.00072 -0.032 -0.035
-0.12 1 0.19 -0.059 0.011 -0.038 0.0058 0.1
0.57 0.19 1 0.037 -0.00061 -0.041 -0.054 0.052
-0.047 -0.059 0.037 1 -0.0047 -0.013 0.042 -0.0097
-0.019 0.011 -0.00061 -0.0047 1 -0.02 -0.12 0.12
-0.00072 -0.038 -0.041 -0.013 -0.02 1 -0.029 -0.025
-0.032 0.0058 -0.054 0.042 -0.12 -0.029 1 -0.045
-0.035 0.1 0.052 -0.0097 0.12 -0.025 -0.045 1
an_Amount_Term_missing
1.0
0.8
0.6
0.4
0.2
0.0

Lo

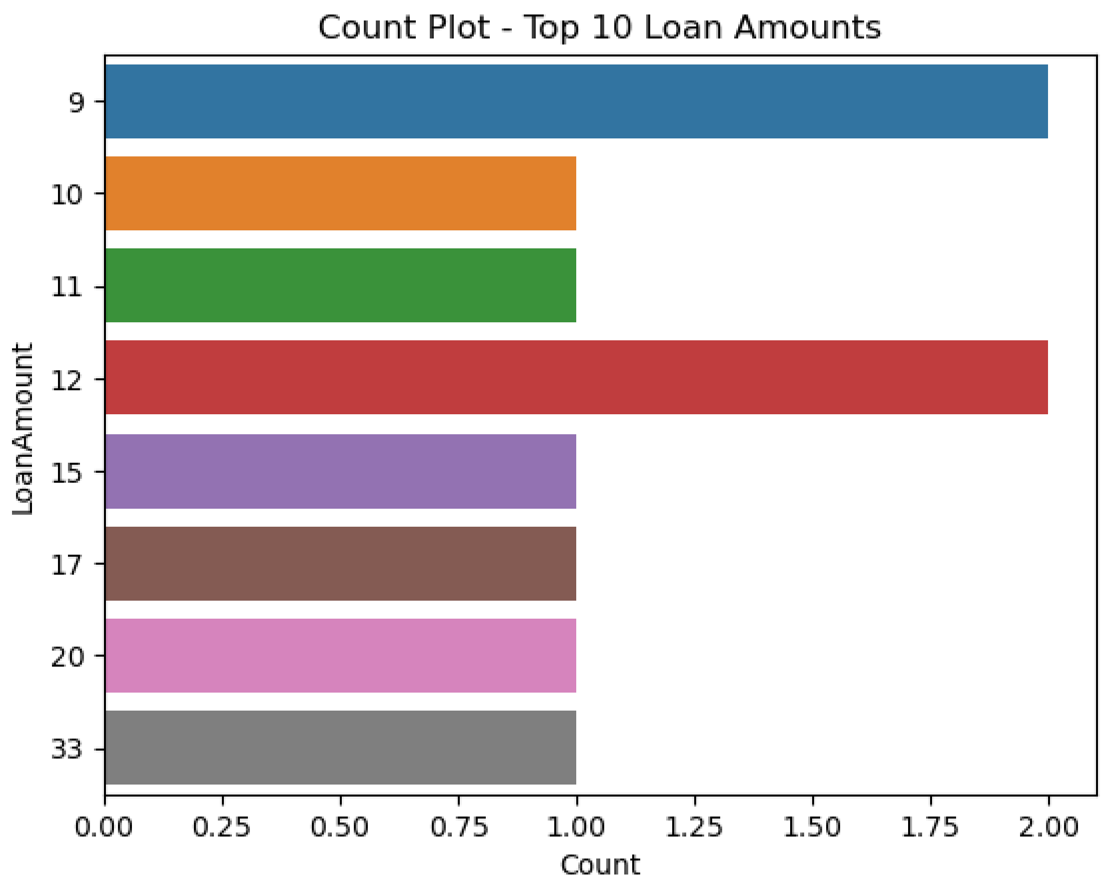
In [457]:
```
# scatter plot
plt.scatter(df['LoanAmount_missing'], df['Loan_Amount_Term_missing'])
plt.xlabel('LoanAmount_missing')
plt.ylabel('Loan_Amount_Term_missing')
plt.title ('Scatter Plot')
plt.show()
```



In [458]:
```
# Box plot
sns.boxplot(y=df['ApplicantIncome'])
plt.ylabel('ApplicantIncome')
plt.xlabel('Count')
plt.title('Box Plot')
plt.show()
```



In [459]:
```
# Select a subset of LoanAmount data for better visualization
subset_loan_amount = df['LoanAmount'].value_counts().head(10)

# Count plot
sns.countplot(y=subset_loan_amount)
plt.ylabel('LoanAmount')
plt.xlabel('Count')
plt.title('Count Plot - Top 10 Loan Amounts')
plt.show()
```



In [460]:
```
px.bar(df, x = 'CoapplicantIncome', y ='LoanAmount', title = 'Bar plot')
```

Feature Engineering

In [461]:
```
loan_df = df.copy()
```

In [462]:
```
loan_df.head()
```

Out[462]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

Separating the data

In [463]:
```
x = loan_df.drop('Loan_Status', axis = 1)
y = loan_df['Loan_Status']
```

In [464]:
```
x.head()
```

Out[464]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [465]:
```
x.isna().sum()
```

Out[465]:
```
Loan_ID                     0
Gender                      0
Married                     0
Dependents                  0
Education                   0
Self_Employed               0
ApplicantIncome             0
CoapplicantIncome           0
LoanAmount                  0
Loan_Amount_Term            0
Credit_History              0
Property_Area               0
LoanAmount_missing          0
Loan_Amount_Term_missing    0
Credit_History_missing      0
dtype: int64
```

In [466]:
```
y
```

Out[466]:
```
0      Y
1      N
2      Y
3      Y
4      Y
      ..
609    Y
610    Y
611    Y
612    Y
613    N
Name: Loan_Status, Length: 614, dtype: object
```

In [467]:
```
# Assuming 'y' is the original label column
y = loan_df['Loan_Status']

# Define label mapping
label_map = {'Y': 1, 'N': 0}

# Map labels to binary values
y_binary = y.map(label_map)

# Print the result
print(y_binary)
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int64
```

In [468]:
```
y = y_binary
```

In [469]:
```
y.shape
```

Out[469]:
```
(614,)
```

In [470]:
```
for label, content in df.items():
    if not pd.api.types.is_numeric_dtype(content):
        print(label)
```

```
Loan_ID
Gender
Married
Dependents
Education
Self_Employed
Property_Area
Loan_Status
```

Whether to use ColumnTransformer or not depends on the specific requirements and complexity of your data preprocessing tasks. Both approaches have their advantages and it's important to consider the context of your project.

Here are some factors to consider when deciding which approach is better for your situation:

Simplicity: If your preprocessing tasks involve applying the same transformation to all columns or require only a few simple transformations, it may be more straightforward to apply them individually without ColumnTransformer.

Complexity: If you have a large number of columns or need to apply different transformations to different subsets of columns, ColumnTransformer can help simplify your code and make it more manageable.

Flexibility: ColumnTransformer offers greater flexibility in handling multiple preprocessing steps. You can easily add, remove, or modify transformations for specific subsets of columns without affecting the rest of the code.

Code readability: ColumnTransformer provides a more structured and concise way to specify preprocessing steps, making your code easier to understand and maintain, especially when dealing with complex transformations.

Future scalability: If you anticipate that your preprocessing requirements may change or expand in the future, using ColumnTransformer can provide a more scalable solution as it allows you to easily incorporate new transformations.

In general, if your preprocessing tasks are simple and involve applying the same transformation to all columns, not using ColumnTransformer may be sufficient. However, if you have more complex preprocessing requirements with different transformations for different subsets of columns, or if you value code organization and scalability, ColumnTransformer can be a better choice.

Ultimately, it's important to assess your specific needs, consider the complexity of your data preprocessing tasks, and evaluate the trade-offs between simplicity, flexibility, and code readability to determine which approach is better suited for your project.

In [471]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

# Select the categorical columns for one-hot encoding
categorical_cols = ['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property

one_hot = OneHotEncoder()
transformer = ColumnTransformer([('one_hot', one_hot, categorical_cols)],
                                remainder='passthrough')

transformed_x = transformer.fit_transform(x)
```

In [472]:
```
# from sklearn.preprocessing import OneHotEncoder

# # Select the categorical columns for one-hot encoding
# categorical_cols = ['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Proper

# # Extract the categorical column values
# df_categorical = x[categorical_cols]

# # Create an instance of the OneHotEncoder
# encoder = OneHotEncoder()

# # Fit and transform the categorical columns
# encoded_data = encoder.fit_transform(df_categorical)

# # Create a new dataframe with the encoded data
# df_encoded = pd.DataFrame(encoded_data.toarray(), columns=encoder.get_feature_names_out(categorical_col
```

In [473]:
```
transformed_x
```

Out[473]:
```
<614x641 sparse matrix of type '<class 'numpy.float64'>'
        with 7092 stored elements in Compressed Sparse Row format>
```

In [474]:
```
x = transformed_x
y = y_binary
```

In [475]:
```
x. shape, y.shape
```

Out[475]:
```
((614, 641), (614,))
```

Splitting our dataset

In [476]:
```
from sklearn.model_selection import train_test_split
```

In [477]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stratify=y, random_state=42)
```

In [478]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[478]:
```
((491, 641), (123, 641), (491,), (123,))
```

Model Training`

In [479]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [480]:
```
models = {
    'LogisticRegression': LogisticRegression(),
    'Mult': MultinomialNB(),
    'Random Forest': RandomForestClassifier(),
    'Decison Tree': DecisionTreeClassifier(),
    'Gradient': GradientBoostingClassifier(),
    'Ada': AdaBoostClassifier(),
    'KNN': KNeighborsClassifier()
}
```

In [481]:
```
# function to fit and score

def fit_and_score(models, x_train, x_test, y_train, y_test):
    np.random.seed(42)
    model_scores = {}

    # Loop through
    for name, model in models.items():
        model.fit(x_train, y_train)

        # evaluate the model
        model_scores[name] = model.score(x_test, y_test)
    return model_scores
```

In [482]:
```
%%time
model_scores = fit_and_score(
    models=models,
    x_train=x_train,
    x_test=x_test,
    y_train=y_train,
    y_test=y_test
)
import warnings
warnings.filterwarnings('ignore')
```

```
Wall time: 1.25 s
```

In [483]:
```
model_scores
```

Out[483]:
```
{'LogisticRegression': 0.8373983739837398,
 'Mult': 0.5040650406504065,
 'Random Forest': 0.8373983739837398,
 'Decison Tree': 0.8211382113821138,
 'Gradient': 0.8536585365853658,
 'Ada': 0.8211382113821138,
 'KNN': 0.6504065040650406}
```

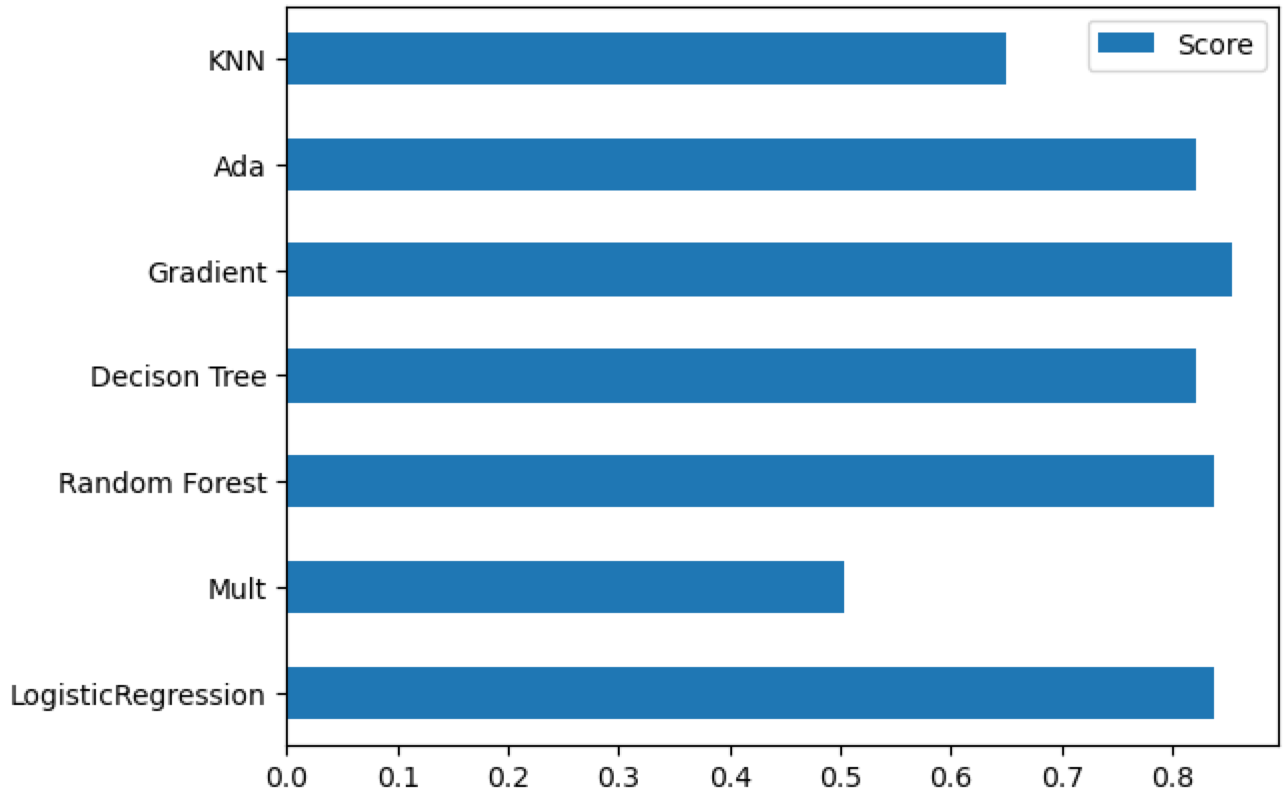
In [484]:
```
# Create a dataframe from the model_scores dictionary
df_scores = pd.DataFrame.from_dict(model_scores, orient='index', columns=['Score'])
```

In [485]:
```
df_scores
```

Out[485]:

	Score
LogisticRegression	0.837398
Mult	0.504065
Random Forest	0.837398
Decison Tree	0.821138
Gradient	0.853659
Ada	0.821138
KNN	0.650407

In [486]:
```
df_scores.plot.barh();
```



Hyperparameter Tunning on Ada

In [487]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV

# Create an AdaBoostClassifier object
ada = AdaBoostClassifier()

# Define the parameter grid with updated values
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.5]
}

# Perform grid search
grid_search = GridSearchCV(ada, param_grid, cv=5)
grid_search.fit(x_train, y_train)

# Print the best parameters and best score
print("Best Parameters: ", grid_search.best_params_)
print("Best Score: ", grid_search.best_score_)
```

```
Best Parameters:  {'learning_rate': 0.01, 'n_estimators': 100}
Best Score:  0.7983714698000413
```

Hyperparameter Tuning on KNN

In [488]:
```
train_scores = []
test_scores = []

# create a list of different values for n_neigbours
neighbors = range(1, 21)

# setup Knn instance

knn = KNeighborsClassifier()
# Loop through
for i in neighbors:
    knn.set_params(n_neighbors = i)
    knn.fit(x_train, y_train)

    train_scores.append(knn.score(x_train, y_train))
    test_scores.append(knn.score(x_test, y_test))
```

In [489]:
```
train_scores
```

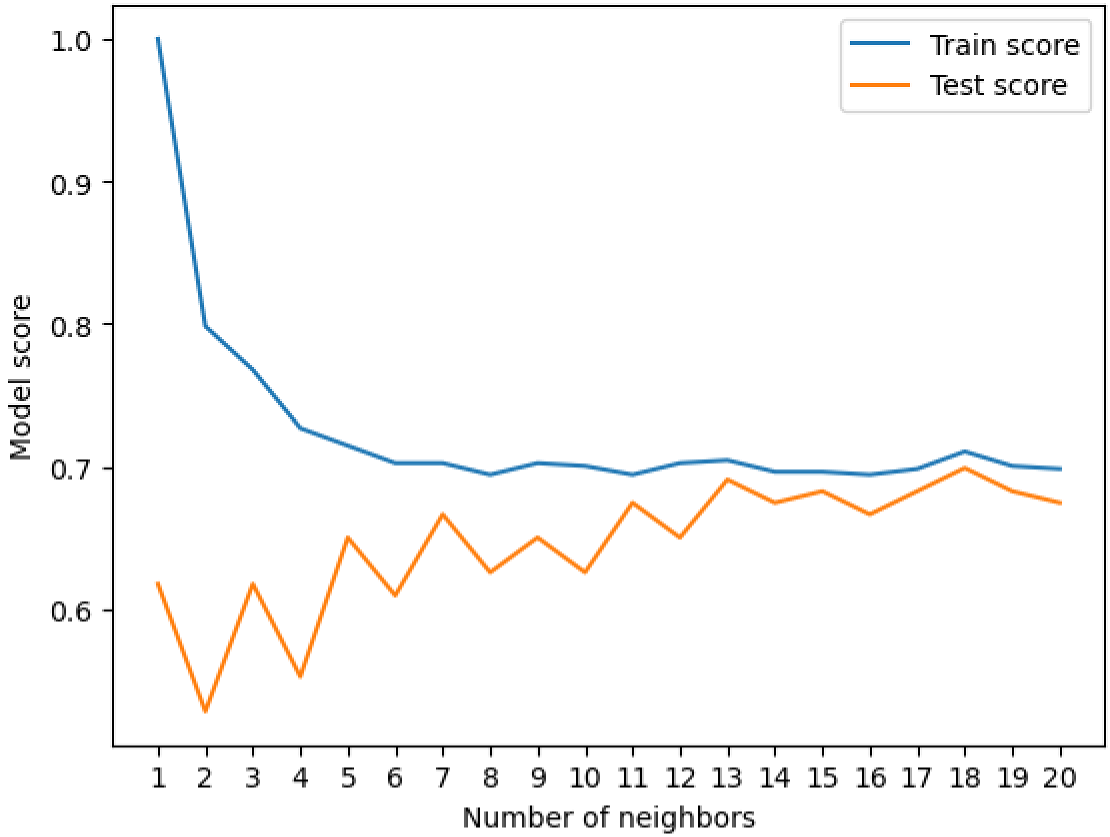
Out[489]:
```
[1.0,
 0.7983706720977597,
 0.7678207739307535,
 0.7270875763747454,
 0.714867617107943,
 0.7026476578411406,
 0.7026476578411406,
 0.6945010183299389,
 0.7026476578411406,
 0.7006109979633401,
 0.6945010183299389,
 0.7026476578411406,
 0.7046843177189409,
 0.6965376782077393,
 0.6965376782077393,
 0.6945010183299389,
 0.6985743380855397,
 0.7107942973523421,
 0.7006109979633401,
 0.6985743380855397]
```

In [490]:
```
test_scores
```

Out[490]:
```
[0.6178861788617886,
 0.5284552845528455,
 0.6178861788617886,
 0.5528455284552846,
 0.6504065040650406,
 0.6097560975609756,
 0.6666666666666666,
 0.6260162601626016,
 0.6504065040650406,
 0.6260162601626016,
 0.6747967479674797,
 0.6504065040650406,
 0.6910569105691057,
 0.6747967479674797,
 0.6829268292682927,
 0.6666666666666666,
 0.6829268292682927,
 0.6991869918699187,
 0.6829268292682927,
 0.6747967479674797]
```

In [491]:
```
plt.plot(neighbors, train_scores, label = 'Train score')
plt.plot(neighbors, test_scores, label = 'Test score')
plt.xticks(np.arange(1, 21, 1))
plt.xlabel('Number of neighbors')
plt.ylabel('Model score')
plt.legend()

print(f'Maximum KNN score on the test data: {max(test_scores) * 100:.2f}%')
```

```
Maximum KNN score on the test data: 69.92%
```



Hyperparaneter Tunning with Randomizedsearhcv on:

- LogisticRegression
- RandomForestClassifier

In [492]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [493]:
```
# grid for logistic
log = {'C': np.logspace(-4, 4, 20),
      'solver': ['liblinear']}

# grid for Randomforest
rf = {'n_estimators': np.arange(10, 1000, 50),
     'max_depth': [None, 3, 5, 10],
     'min_samples_split': np.arange(2, 20, 2),
     'min_samples_leaf': np.arange(1, 20, 2)}
```

Hyperparameter grids setup successful

In [494]:
```
# Tune Logistic
np.random.seed(42)

# for log
rs_log = RandomizedSearchCV(LogisticRegression(),
                           param_distributions=log,
                           cv = 5,
                           n_iter=20,
                           verbose=True)

# fit model for Log
rs_log.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[494]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_iter=20,
                   param_distributions={'C': array([1.00000000e-04, 2.63665090e-04, 6.95192796e-04, 1.83
298071e-03,
       4.83293024e-03, 1.27427499e-02, 3.35981829e-02, 8.85866790e-02,
       2.33572147e-01, 6.15848211e-01, 1.62377674e+00, 4.28133240e+00,
       1.12883789e+01, 2.97635144e+01, 7.84759970e+01, 2.06913808e+02,
       5.45559478e+02, 1.43844989e+03, 3.79269019e+03, 1.00000000e+04]),
                                        'solver': ['liblinear']},
                   verbose=True)
```

In [495]:
```
rs_log.best_params_
```

Out[495]:
```
{'solver': 'liblinear', 'C': 29.763514416313132}
```

In [496]:
```
rs_log.score(x_test, y_test)
```

Out[496]:
```
0.8617886178861789
```

In [498]:
```
# Randomizedsearchcv for RandomForest

np.random.seed(42)
rs_rf = RandomizedSearchCV(RandomForestClassifier(),
                          param_distributions= rf,
                          cv=5,
                          n_iter=20,
                          verbose=True)
rs_rf.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[498]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=20,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'min_samples_leaf': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 1
9]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 1
8]),
                                        'n_estimators': array([ 10,  60, 110, 160, 210, 260, 310, 360, 4
10, 460, 510, 560, 610,
       660, 710, 760, 810, 860, 910, 960])},
                   verbose=True)
```

In [499]:
```
rs_rf.score(x_test, y_test)
```

Out[499]:
```
0.8536585365853658
```

From the score so far, LogisticsRegression provided high score of: 0.8617886178861789

In [500]:
```
model_scores
```

Out[500]:
```
{'LogisticRegression': 0.8373983739837398,
 'Mult': 0.5040650406504065,
 'Random Forest': 0.8373983739837398,
 'Decison Tree': 0.8211382113821138,
 'Gradient': 0.8536585365853658,
 'Ada': 0.8211382113821138,
 'KNN': 0.6504065040650406}
```

Hyperparameter using GridSearchCV on LogisticRegression

In [504]:
```
#  grid for logistic
logs = {'C': np.logspace(-4, 4, 40),
      'solver': ['liblinear']}

log_reg =GridSearchCV(LogisticRegression(),
                     param_grid= logs,
                     cv=5,
                     verbose=True)
log_reg.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
```

Out[504]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.00000000e-04, 1.60371874e-04, 2.57191381e-04, 4.12462638e-04,
       6.61474064e-04, 1.06081836e-03, 1.70125428e-03, 2.72833338e-03,
       4.37547938e-03, 7.01703829e-03, 1.12533558e-02, 1.80472177e-02,
       2.89426612e-02, 4.64158883e-02, 7.44380301e-02, 1.19377664e-01,
       1.91448198e-01, 3.07029063e-01, 4.923882...652287e-01,
       1.26638017e+00, 2.03091762e+00, 3.25702066e+00, 5.22334507e+00,
       8.37677640e+00, 1.34339933e+01, 2.15443469e+01, 3.45510729e+01,
       5.54102033e+01, 8.88623816e+01, 1.42510267e+02, 2.28546386e+02,
       3.66524124e+02, 5.87801607e+02, 9.42668455e+02, 1.51177507e+03,
       2.42446202e+03, 3.88815518e+03, 6.23550734e+03, 1.00000000e+04]),
                         'solver': ['liblinear']},
             verbose=True)
```

In [505]:
```
log_reg.best_params_
```

Out[505]:
```
{'C': 0.30702906297578497, 'solver': 'liblinear'}
```

In [506]:
```
log_reg.score(x_test, y_test)
```

Out[506]:
```
0.8617886178861789
```

i am getting the same score as the previous in RandomSearchcv

Evaluating our tunned machine learning classifier, beyond accuracy

let's make prediction before further evaluation

Y = 1 (yes)

N = 0 (No)

In [510]:
```
y_preds = log_reg.predict(x_test)
```

In [511]:
```
y_preds
```

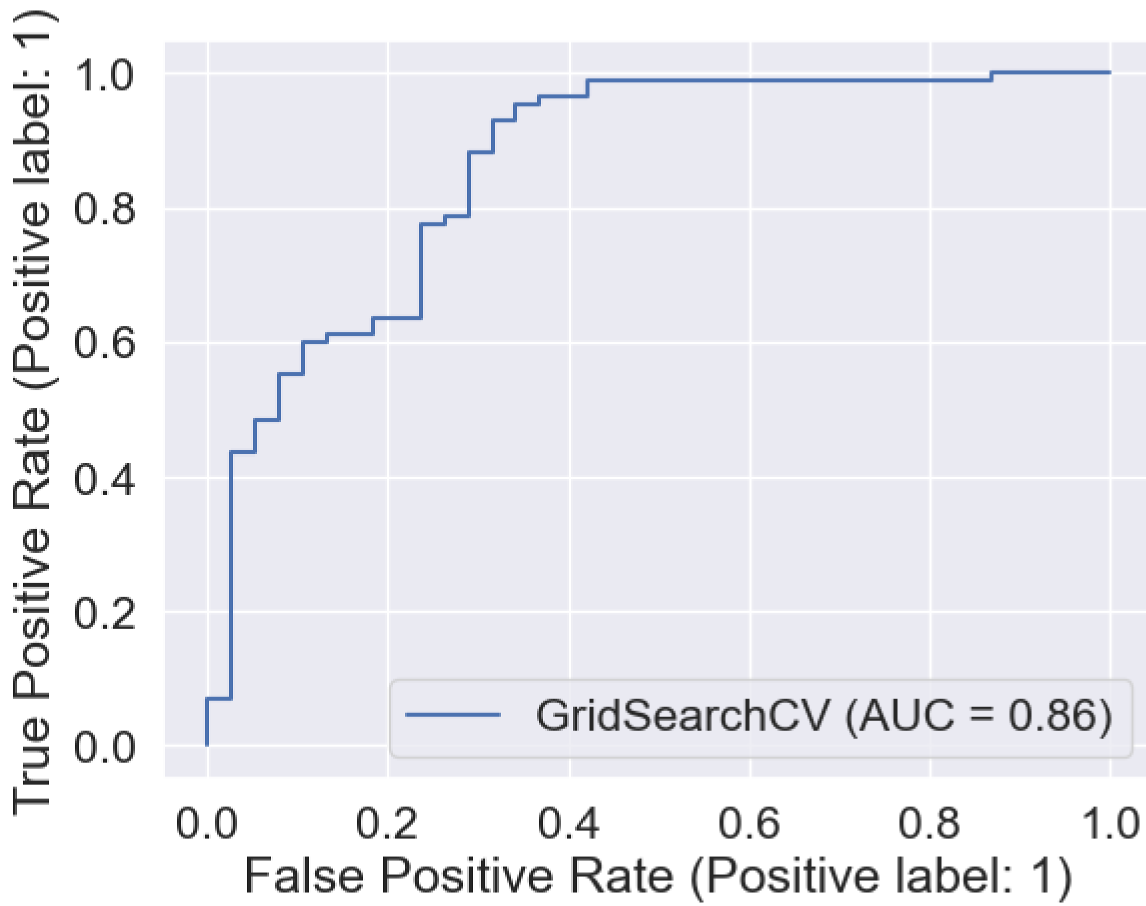
Out[511]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0], dtype=int64)
```

In [515]:
```
#  Model evaluation dependencies
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import plot_roc_curve
import seaborn as sns
```

Evaluating:

```
- Roc and AUC
- Confusion matrix
- classification report
- precision
- recall
- fi-score
```

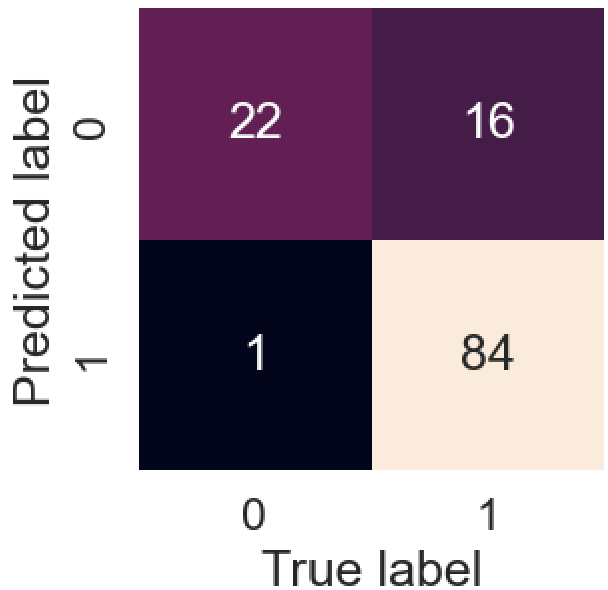
In [517]:
```
# plot ROC curve and calculate AUC metrics
plot_roc_curve(log_reg, x_test, y_test);
```



In [514]:
```
# confusion matrix
confusion_matrix(y_test, y_preds)
```

Out[514]:
```
array([[22, 16],
       [ 1, 84]], dtype=int64)
```

In [516]:
```
sns.set(font_scale = 1.5)

def plot_conf_mat(y_test, y_preds):
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, y_preds),
                    annot=True,
                    cbar=False)
    plt.xlabel('True label')
    plt.ylabel('Predicted label')
plot_conf_mat(y_test, y_preds)
```



In [519]:
```
# classification report

print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.96      0.58      0.72        38
           1       0.84      0.99      0.91        85

    accuracy                           0.86       123
   macro avg       0.90      0.78      0.81       123
weighted avg       0.88      0.86      0.85       123
```

In [520]:
```
from sklearn.model_selection import cross_val_score
```

calculate evaluation metrics using cross-validation

calculating accuracy, precision, recall, and f1-score of our model using cross validation

In [521]:
```
log_reg.best_params_
```

Out[521]:
```
{'C': 0.30702906297578497, 'solver': 'liblinear'}
```

In [522]:
```
# new classifier with best params

lf = LogisticRegression(C=0.30702906297578497,
                       solver='liblinear')
```

In [528]:
```
# cross-validated accuracy
cv_acc = cross_val_score(lf,
                    x, y,
                    cv=5,
                    scoring='accuracy')
cv1 = (np.mean(cv_acc))
cv1
```

Out[528]:
```
0.8061975209916034
```

In [530]:
```
# cross-validated precision
cv_precission = cross_val_score(lf,
                    x, y,
                    cv=5,
                    scoring='precision')
cv2 = np.mean(cv_precission)
cv2
```

Out[530]: 0.790255923083376

In [531]:
```
# cross-validated recall
cv_recall = cross_val_score(lf,
                    x, y,
                    cv=5,
                    scoring='recall')
cv3 = np.mean(cv_recall)
cv3
```

Out[531]: 0.9786834733893558

In [532]:
```
# cross-validated f1
cv_f1 = cross_val_score(lf,
                    x, y,
                    cv=5,
                    scoring='f1')
cv4 = np.mean(cv_f1)
cv4
```

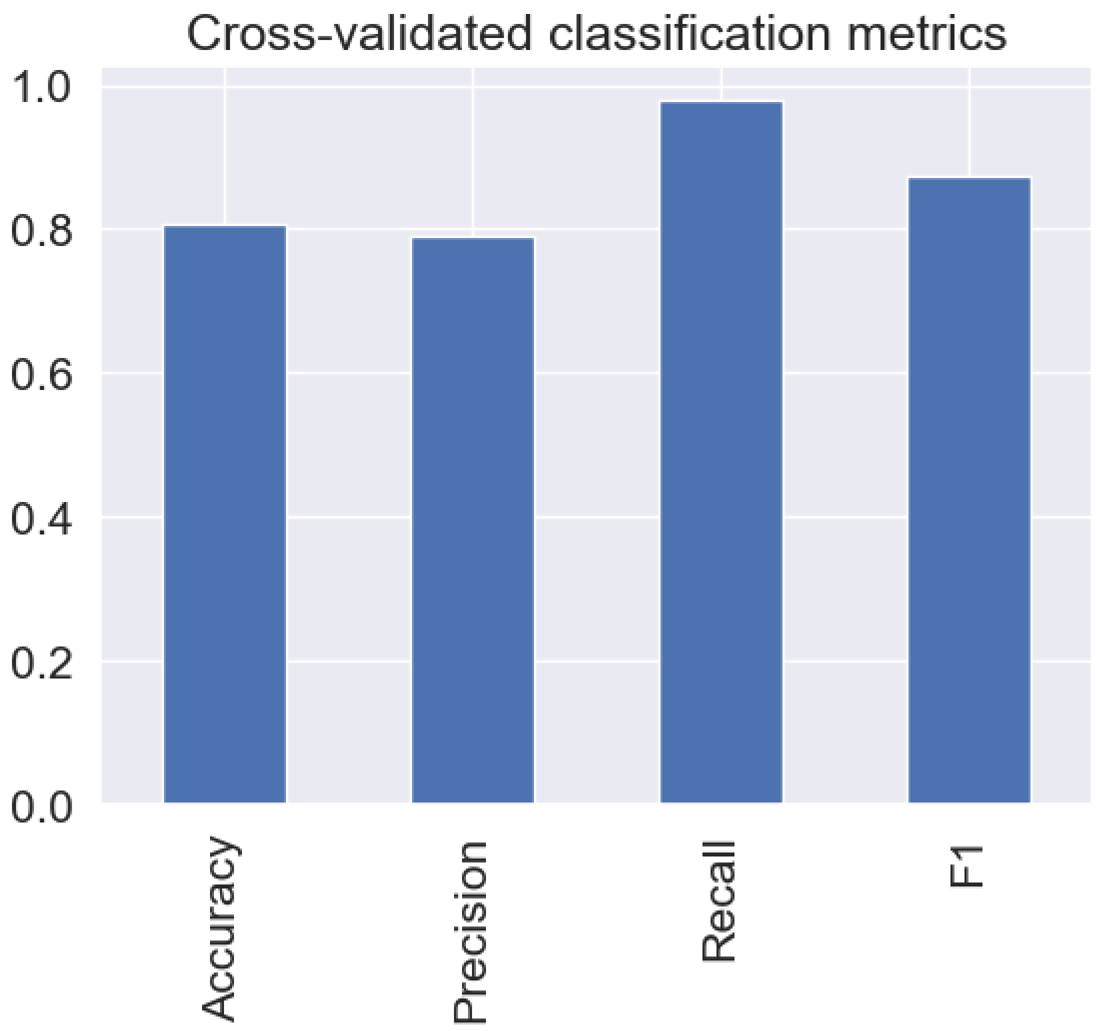
Out[532]: 0.8743131645183372

In [537]:
```
# vitualize cross-validated-metrics
cv_metrics = pd.DataFrame({'Accuracy': cv1,
                           'Precision': cv2,
                           'Recall': cv3,
                           'F1': cv4},
                          index=[0])
cv_metrics
```

Out[537]:

	Accuracy	Precision	Recall	F1
0	0.806198	0.790256	0.978683	0.874313

In [542]:
```
# vitualize cross-validated-metrics
cv_metrics = pd.DataFrame({'Accuracy': cv1,
                           'Precision': cv2,
                           'Recall': cv3,
                           'F1': cv4},
                          index=[0])
cv_metrics.T.plot.bar(title = 'Cross-validated classification metrics',
                      legend = False);
```



Feature importance

let's find trhe feature importance of LogisticRegression, since that is the model we chose

In [544]:
```
log_reg.best_params_


clf = LogisticRegression(C=0.30702906297578497,
                         solver='liblinear')
clf.fit(x_train, y_train);
```

In [545]:
```
# checking coef__
clf.coef_
```

```
         0.00000000e+00, -2.00455011e-01,  6.71799744e-02,
         0.00000000e+00,  2.84922643e-02,  7.99886861e-02,
         6.01714803e-02, -1.92041550e-01,  4.58139166e-02,
        -2.00310800e-01,  6.38414996e-02,  5.00794350e-02,
         0.00000000e+00,  4.11872878e-02,  0.00000000e+00,
        -4.34369077e-02, -1.48785002e-01,  2.92241820e-02,
        -2.05400058e-01,  0.00000000e+00,  0.00000000e+00,
        -5.24278704e-02,  6.97829713e-02,  0.00000000e+00,
         5.27370756e-02,  0.00000000e+00,  0.00000000e+00,
        -1.81759636e-01,  0.00000000e+00,  0.00000000e+00,
        -2.07786379e-01,  0.00000000e+00,  8.89950044e-02,
         3.91002620e-02, -1.70695622e-01,  6.01404181e-02,
         1.64131701e-01, -1.83599065e-01,  8.30222881e-02,
         3.71924206e-02,  5.83474185e-02,  6.47227451e-02,
         0.00000000e+00,  7.13538792e-02, -2.04597345e-01,
        -2.08126319e-01, -9.81436818e-02,  7.07078393e-02,
         7.80132610e-02,  0.00000000e+00,  4.48155766e-02,
         0.00000000e+00,  3.79258479e-02, -1.14144745e-01,
         0.00000000e+00, -6.97584391e-02,  0.00000000e+00,
```

In [547]:
```
loan_df.head()
```

Out[547]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

In [548]:
```
# match coef's of features to colums
feature_dict = dict(zip(loan_df.columns, list(clf.coef_[0])))
feature_dict
```

Out[548]:
```
{'Loan_ID': 0.08894117611747393,
 'Gender': -0.17880812620354344,
 'Married': 0.0,
 'Dependents': 0.07517372728073024,
 'Education': 0.07346917711024831,
 'Self_Employed': 0.057700437282724774,
 'ApplicantIncome': 0.07247110015717592,
 'CoapplicantIncome': 0.0,
 'LoanAmount': 0.04563975967885231,
 'Loan_Amount_Term': -0.18175090316976342,
 'Credit_History': 0.04297780083190181,
 'Property_Area': 0.0,
 'Loan_Status': 0.05994033579537075,
 'LoanAmount_missing': -0.1634509427168705,
 'Loan_Amount_Term_missing': 0.03439884963762259,
 'Credit_History_missing': 0.07295388333862406}
```

In [549]:
```
# visualize feature importance

feature_df = pd.DataFrame(feature_dict, index=[0])
feature_df
```

Out[549]:

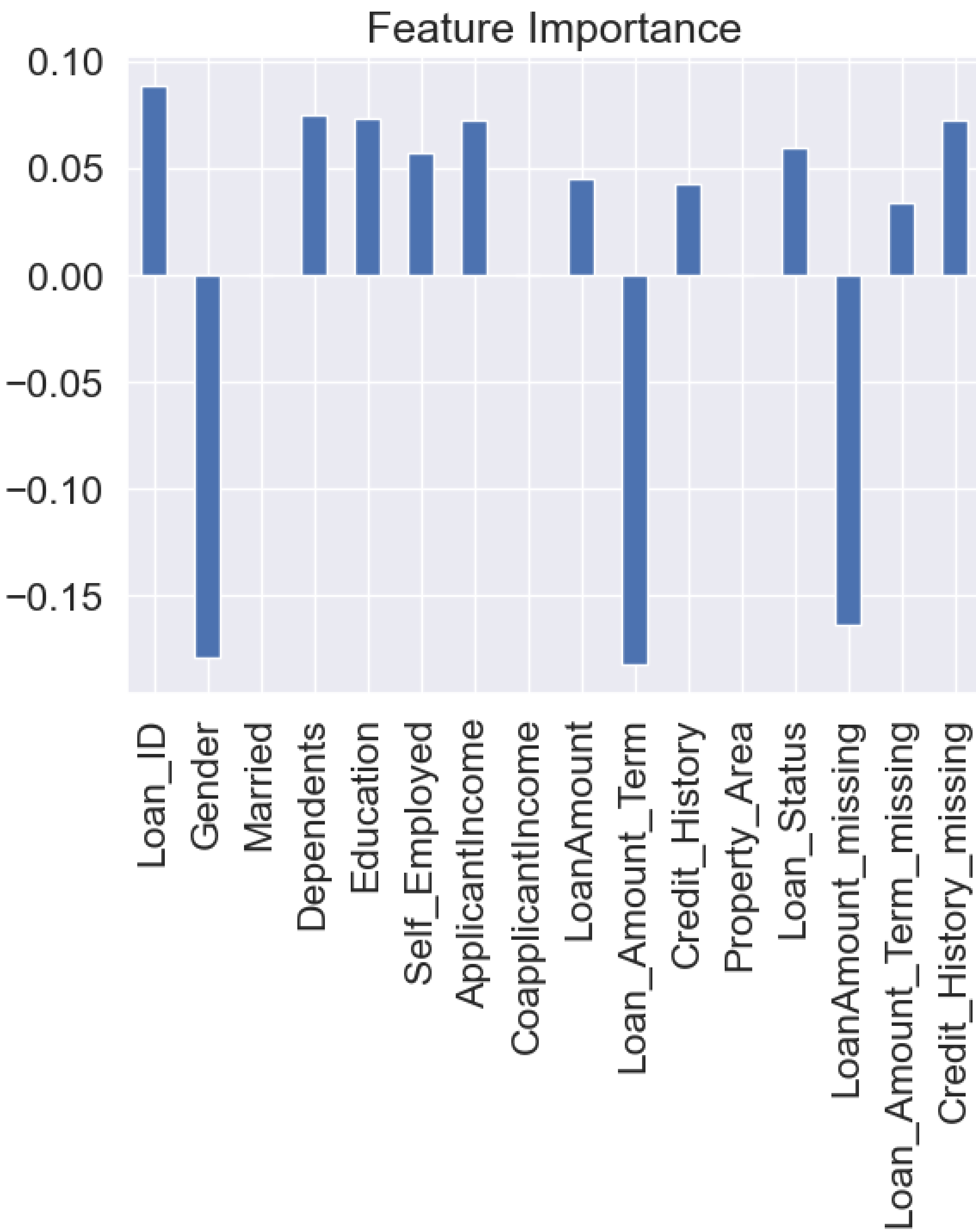
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amour
0	0.088941	-0.178808	0.0	0.075174	0.073469	0.0577	0.072471	0.0	0.04564	-0.

In [550]:
```
feature_df.T
```

Out[550]:

	0
Loan_ID	0.088941
Gender	-0.178808
Married	0.000000
Dependents	0.075174
Education	0.073469
Self_Employed	0.057700
ApplicantIncome	0.072471
CoapplicantIncome	0.000000
LoanAmount	0.045640
Loan_Amount_Term	-0.181751
Credit_History	0.042978
Property_Area	0.000000
Loan_Status	0.059940
LoanAmount_missing	-0.163451
Loan_Amount_Term_missing	0.034399
Credit_History_missing	0.072954

In [552]:
```
feature_df.T.plot.bar(title = 'Feature Importance', legend = False);
```


Feature Importance
0.10
0.05
0.00
−0.05
−0.10
−0.15
Loan_ID
Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
LoanAmount_missing
Loan_Amount_Term_missing
Credit_History_missing

In [555]:
```
pd.crosstab(loan_df['Loan_ID'], loan_df['Dependents'])
```

Out[555]:

Dependents	0	1	2	3+
Loan_ID				
LP001002	0	1	0	0	0
LP001003	0	0	1	0	0
LP001005	0	1	0	0	0
LP001006	0	1	0	0	0
LP001008	0	1	0	0	0
...	...	...	...	...	...
LP002978	0	1	0	0	0
LP002979	0	0	0	0	1
LP002983	0	0	1	0	0
LP002984	0	0	0	1	0
LP002990	0	1	0	0	0

614 rows × 5 columns

In [ ]: